[**Lecture 1**](https://cs50.harvard.edu/python/2022/notes/1/#lecture-1)

* [Conditionals](https://cs50.harvard.edu/python/2022/notes/1/#conditionals)
* [if Statements](https://cs50.harvard.edu/python/2022/notes/1/#if-statements)
* [Control Flow, elif, and else](https://cs50.harvard.edu/python/2022/notes/1/#control-flow-elif-and-else)
* [or](https://cs50.harvard.edu/python/2022/notes/1/#or)
* [and](https://cs50.harvard.edu/python/2022/notes/1/#and)
* [Modulo](https://cs50.harvard.edu/python/2022/notes/1/#modulo)
* [Creating Our Own Parity Function](https://cs50.harvard.edu/python/2022/notes/1/#creating-our-own-parity-function)
* [Pythonic](https://cs50.harvard.edu/python/2022/notes/1/#pythonic)
* [match](https://cs50.harvard.edu/python/2022/notes/1/#match)
  + [Summing Up](https://cs50.harvard.edu/python/2022/notes/1/#summing-up)

[**Conditionals**](https://cs50.harvard.edu/python/2022/notes/1/#conditionals)

* Conditionals allow you, the programmer, to allow your program to make decisions: As if your program has the choice between taking the left-hand road or the right-hand road based upon certain conditions.
* Built within Python are a set of “operators” that are used to ask mathematical questions.
* > and < symbols are probably quite familiar to you.
* >= denotes “greater than or equal to.”
* <= denotes “less than or equal to.”
* == denotes “equals, though do notice the double equal sign! A single equal sign would assign a value. Double equal signs are used to compare variables.
* != denotes “not equal to.
* Conditional statements compare a left-hand term to a right-hand term.

[**if Statements**](https://cs50.harvard.edu/python/2022/notes/1/#if-statements)

* In your terminal window, type code compare.py. This will create a brand new file called “compare.”
* In the text editor window, begin with the following:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x < y:
* print("x is less than y")

Notice how your program takes the input of the user for both x and y, casting them as integers and saving them into their respective x and y variables. Then, the if statement compares x and y. If the condition of x < y is met, the print statement is executed.

* if statements use bool or boolean values (true or false) to decide whether or not to execute. If the statement of x > y is true, the compiler will register it as true and execute the code.

[**Control Flow, elif, and else**](https://cs50.harvard.edu/python/2022/notes/1/#control-flow-elif-and-else)

* Further revise your code as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x < y:
* print("x is less than y")
* **if** x > y:
* print("x is greater than y")
* **if** x == y:
* print("x is equal to y")

Notice how you are providing a series of if statements. First, the first if statement is evaluated. Then, the second if statement runs its evaluation. Finally, the last if statement runs its evaluation. This flow of decisions is called “control flow.”

* Our code can be represented as follows:

True

True

True

False

False

False

start

x < y

"x is less than y"

x > y

"x is greater than y"

x == y

"x is equal to y"

stop

* This program can be improved by not asking three consecutive questions. After all, not all three questions can have an outcome of true! Revise your program as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x < y:
* print("x is less than y")
* **elif** x > y:
* print("x is greater than y")
* **elif** x == y:
* print("x is equal to y")

Notice how the use of elif allows the program to make fewer decisions. First, the if statement is evaluated. If this statement is found to be true, all the elif statements will not be run at all. However, if the if statement is evaluated and found to be false, the first elif will be evaluated. If this is true, it will not run the final evaluation.

* Our code can be represented as follows:

True

False

True

False

True

False

start

x < y

"x is less than y"

x > y

"x is greater than y"

x == y

"x is equal to y"

stop

* While your computer may not notice a difference speed-wise between our first program and this revised program, consider how an online server running billions or trillions of these types of calculations each day could definitely be impacted by such a small coding decision.
* There is one final improvement we can make to our program. Notice how logically elif x == y is not a necessary evaluation to run. After all, if logically x is not less than y AND x is not greater than y, x MUST equal y. Therefore, we don’t have to run elif x == y. We can create a “catch-all,” default outcome using an else statement. We can revise as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x < y:
* print("x is less than y")
* **elif** x > y:
* print("x is greater than y")
* **else**:
* print("x is equal to y")

Notice how the relative complexity of this program has decreased through our revision.

* Our code can be represented as follows:

True

False

True

False

start

x < y

"x is less than y"

x > y

"x is greater than y"

"x is equal to y"

stop

[**or**](https://cs50.harvard.edu/python/2022/notes/1/#or)

* or allows your program to decide between one or more alternatives. For example, we could further edit our program as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x < y **or** x > y:
* print("x is not equal to y")
* **else**:
* print("x is equal to y")

Notice that the result of our program is the same, but the complexity is decreased. The efficiency of our code is increased.

* At this point, our code is pretty great. However, could the design be further improved? We could further edit our code as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x != y:
* print("x is not equal to y")
* **else**:
* print("x is equal to y")

Notice how we removed the or entirely and simply asked, “Is x not equal to y?” We ask one and only one question. Very efficient!

* For the purpose of illustration, we could also change our code as follows:
* x = int(input("What's x? "))
* y = int(input("What's y? "))
* **if** x == y:
* print("x is equal to y")
* **else**:
* print("x is not equal to y")

Notice that the == operator evaluates if what is on the left and right are equal to one another. The use of double equal signs is very important. If you use only one equal sign, an error will likely be thrown by the compiler.

* Our code can be illustrated as follows:

True

False

start

x == y

"x is equal to y"

"x is not equal to y"

stop

[**and**](https://cs50.harvard.edu/python/2022/notes/1/#and)

* Similar to or, and can be used within conditional statements.
* Execute in the terminal window code grade.py. Start your new program as follows:
* score = int(input("Score: "))
* **if** score >= 90 **and** score <= 100:
* print("Grade: A")
* **elif** score >=80 **and** score < 90:
* print("Grade: B")
* **elif** score >=70 **and** score < 80:
* print("Grade: C")
* **elif** score >=60 **and** score < 70:
* print("Grade: D")
* **else**:
* print("Grade: F")

Notice that by executing python grade.py, you will be able to input a score and get a grade. However, notice how there is potential for bugs.

* Typically, we do not want to ever trust our users to input the correct information. We could improve our code as follows:
* score = int(input("Score: "))
* **if** 90 <= score <= 100:
* print("Grade: A")
* **elif** 80 <= score < 90:
* print("Grade: B")
* **elif** 70 <= score < 80:
* print("Grade: C")
* **elif** 60 <= score < 70:
* print("Grade: D")
* **else**:
* print("Grade: F")

Notice how Python allows you to chain together the operators and conditions in a way quite uncommon to other programming languages.

* Still, we can further improve our program:
* score = int(input("Score: "))
* **if** score >= 90:
* print("Grade: A")
* **elif** score >= 80:
* print("Grade: B")
* **elif** score >= 70:
* print("Grade: C")
* **elif** score >= 60:
* print("Grade: D")
* **else**:
* print("Grade: F")

Notice how the program is improved by asking fewer questions. This makes our program easier to read and far more maintainable in the future.

* You can learn more in Python’s documentation on [control flow](https://docs.python.org/3/tutorial/controlflow.html).

[**Modulo**](https://cs50.harvard.edu/python/2022/notes/1/#modulo)

* In mathematics, parity refers to whether a number is either even or odd.
* The modulo % operator in programming allows one to see if two numbers divide evenly or divide and have a remainder.
* For example, 4 % 2 would result in zero, because it evenly divides. However, 3 % 2 does not divide evenly and would result in a number other than zero!
* In the terminal window, create a new program by typing code parity.py. In the text editor window, type your code as follows:
* x = int(input("What's x? "))
* **if** x % 2 == 0:
* print("Even")
* **else**:
* print("Odd")

Notice how our users can type in any number 1 or greater to see if it is even or odd.

[**Creating Our Own Parity Function**](https://cs50.harvard.edu/python/2022/notes/1/#creating-our-own-parity-function)

* As discussed in Lecture 0, you will find it useful to create a function of your own!
* We can create our own function to check whether a number is even or odd. Adjust your code as follows:
* **def** main():
* x = int(input("What's x? "))
* **if** is\_even(x):
* print("Even")
* **else**:
* print("Odd")
* **def** is\_even(n):
* **if** n % 2 == 0:
* **return** True
* **else**:
* **return** False
* main()

Notice that our if statement is\_even(x) works even though there is no operator there. This is because our function returns a bool (or boolean), true or false, back to the main function. The if statement simply evaluates whether or not is\_even of x is true or false.

[**Pythonic**](https://cs50.harvard.edu/python/2022/notes/1/#pythonic)

* In the programming world, there are types of programming that are called “Pythonic” in nature. That is, there are ways to program that are sometimes only seen in Python programming. Consider the following revision to our program:
* **def** main():
* x = int(input("What's x? "))
* **if** is\_even(x):
* print("Even")
* **else**:
* print("Odd")
* **def** is\_even(n):
* **return** True **if** n % 2 == 0 **else** False
* main()

Notice that this return statement in our code is almost like a sentence in English. This is a unique way of coding only seen in Python.

* We can further revise our code and make it more and more readable:
* **def** main():
* x = int(input("What's x? "))
* **if** is\_even(x):
* print("Even")
* **else**:
* print("Odd")
* **def** is\_even(n):
* **return** n % 2 == 0
* main()

Notice that the program will evaluate what is happening within the n % 2 == 0 as either true or false and simply return that to the main function.

[**match**](https://cs50.harvard.edu/python/2022/notes/1/#match)

* Similar to if, elif, and else statements, match statements can be used to conditionally run code that matches certain values.
* Consider the following program:
* name = input("What's your name? ")
* **if** name == "Harry":
* print("Gryffindor")
* **elif** name == "Hermione":
* print("Gryffindor")
* **elif** name == "Ron":
* print("Gryffindor")
* **elif** name == "Draco":
* print("Slytherin")
* **else**:
* print("Who?")

Notice the first three conditional statements print the same response.

* We can improve this code slightly with the use of the or keyword:
* name = input("What's your name? ")
* **if** name == "Harry" **or** name == "Hermione" **or** name == "Ron":
* print("Gryffindor")
* **elif** name == "Draco":
* print("Slytherin")
* **else**:
* print("Who?")

Notice the number of elif statements has decreased, improving the readability of our code.

* Alternatively, we can use match statements to map names to houses. Consider the following code:
* name = input("What's your name? ")
* match name:
* case "Harry":
* print("Gryffindor")
* case "Hermione":
* print("Gryffindor")
* case "Ron":
* print("Gryffindor")
* case "Draco":
* print("Slytherin")
* case \_:
* print("Who?")

Notice the use of the \_ symbol in the last case. This will match with any input, resulting in similar behavior as an else statement.

* A match statement compares the value following the match keyword with each of the values following the case keywords. In the event a match is found, the respective indented code section is executed, and the program stops the matching.
* We can improve the code:
* name = input("What's your name? ")
* match name:
* case "Harry" | "Hermione" | "Ron":
* print("Gryffindor")
* case "Draco":
* print("Slytherin")
* case \_:
* print("Who?")

Notice, the use of the single vertical bar |. Much like the or keyword, this allows us to check for multiple values in the same case statement.

[**Summing Up**](https://cs50.harvard.edu/python/2022/notes/1/#summing-up)

You now have the power within Python to use conditional statements to ask questions and have your program take action accordingly. In this lecture, we discussed…

* Conditionals;
* if Statements;
* Control flow, elif, and else;
* or;
* and;
* Modulo;
* Creating your own function;
* Pythonic coding;
* and match.

[MUSIC PLAYING]

DAVID MALAN: This is CS50's Introduction to Programming with Python.

My name is David Malan, and this week we focus on conditionals.

Conditionals, or conditional statements, in Python and in other languages,

are this ability to ask questions and answer those questions, in order

to decide do you want to execute this line of code?

Or this line of code?

Or this other line of code instead?

They allow you to take the proverbial forks in the road,

within your own code, logically.

So how might we go about making some of these decisions?

Well, it turns out that Python comes with a lot of built-in syntax.

For instance, here are just some of the symbols

you can use in Python to ask questions.

Admittedly, mathematical questions, but we'll start there, if only

to keep the examples simple early on.

This first symbol, as you might know from math, represents greater than.

The second symbol might not look too familiar, because we usually write it

all as one thing on a piece of paper.

But on a keyboard, if you want to say greater than or equal to,

you'd use this symbol instead.

This, of course, means less than.

This means less than or equal to.

And this one's a bit of a curiosity.

We've seen, in our look at functions and variables,

how we were able to assign values to variables using a single equal sign.

But that equal sign didn't represent equality.

It represented assignment, from right to left.

That's great, because it solved that problem.

But it left us in a bit of a bind, because how do we now

compare two things, left and right?

Well, in Python, and in many languages, you actually use two equal sides.

So two equal signs represents equality, comparing the thing

on the left and the right.

One equal sign, as always, represents assignment,

copying the thing from the right to the left.

Lastly, this last symbol represents not equal to.

So the exclamation point, or bang, followed by an equal sign,

means not equal to some value next to it.

Well, to ask the questions using these symbols, or any others,

we're going to need another keyword in Python.

And that keyword, quite simply, as in English, is if.

You can ask questions in Python code along the lines of,

if the answer to this question is true, then go ahead

and execute this code for me.

So let's go ahead and write some of these examples here.

I'm going to go over to VS Code.

And let's go ahead and create a program first,

called compare.py, the goal of which is simply

to write code that compares values and makes decisions based on those values.

Let's go ahead and type code of compare.py,

in order to create a brand new file called compare,

in which we'll start to express some of this logic.

Well, what do we want to compare?

Suppose we want to compare, for the sake of discussion,

just a couple of integers.

But we'd like those integers to come from the user,

so that we can make decisions based on numbers

we don't know the values of in advance.

Well, let's go ahead and do this.

As we've done in the past, let's declare it a variable, like x.

Let's assign it equal to the return value of the int function,

and pass to the int function the return value of the input function,

asking the user a question like, what's x, question

mark, as we've done in the past.

Let's do this one more time with y, asking the user for the value of y.

And, again, converting that, ultimately, to an int, as well.

So with this amount of the story, we have two variables, x and y,

each of which has values.

And ideally, we should be able to now compare these values.

So suppose I want to make a decision based on the values of these variables.

I'm going to use the keyword if.

And I'm going to use some of those mathematical symbols

to actually ask the question itself.

So how about this, if x is less than y, then let's go ahead and just

print as much out.

Quote, unquote x is less than y.

So this isn't a very interesting program yet.

I'm literally just stating the obvious, based on the math.

But it's allowing me to now introduce some new syntax.

And exactly what is the syntax?

Well, it's this-- not just the keyword if, which

I've added here at the start of line four,

but then I asked my question here, x less than y.

x is one variable on the left, y is one variable on the right.

And, of course, the less than sign is expressing the mathematical question

I have.

What I've highlighted here is technically

called a Boolean expression.

A Boolean expression, named after a mathematician named Bool,

is simply a question that has a yes or no answer, or technically,

a true or false answer.

And that's nice because if there's only two possible answers,

it's very easy for me, and in turn the computer, to make a decision--

do this, or don't do this thing.

Now notice, if you come from other languages,

you might notice that I have not typed any parentheses.

They are not, in fact, necessary, at least in this case, in Python,

but I have typed a colon at the end of the line.

And even more importantly, at the next line

I have begun my line with some indentation,

hitting the space bar four times, or just hitting Tab once,

which will automatically be converted to the same.

That indentation is what tells Python that line five should only

be executed if the answer to line four's question is, in fact, true.

So if x is less than y, that phrase will be printed thereafter.

Well, let's add a few more lines of code.

How about another question?

If x is greater than y, then let's go ahead and print that.

x is greater than y.

And let's do one final question, if x equals y, then-- wait a minute.

What have I done wrong here?

A good eye here.

I don't want to assign y to x.

If x equals equals y is how I express equality, let's go ahead

and print out x is equal to y.

So I now have three conditions, if you will,

one question asking x less than y, one asking x greater than y,

one asking x equals equals y.

Let's run the code.

Well, down here in my terminal window I'm

going to run Python of compare.py and hit Enter.

What's x?

Let's go with one.

What's y?

Let's go with two.

This should, of course, execute that first line of code

and tell me, indeed, that x is less than y.

Exactly as I would expect there.

Well, what just happened, though, in code?

Well, let's take a look, perhaps, at this same code visually,

particularly if you're a more visual learner, this, I dare say,

is what just happened.

So what we're looking at here is a flow chart.

It's a diagram of this program's logic.

And more technically, it shows the program's control flow.

That is, the ability of you, in code, to control

the flow of a program, generally from top to bottom.

In fact, let me go ahead and zoom in on the top of this flow chart.

And you'll see an oval at the very top that says, quite literally, start.

That is, irrespective of what shape or layout

the diagram is, where your own thinking and logic

should start when trying to wrap your mind around this program.

Notice that there's an arrow from start to this diamond shape.

And inside of that diamond is a question, a Boolean expression,

x less than y.

And this shape just means, based on the answer to that question,

go left or go right.

Specifically, go left if the answer is true,

or go right if the answer is false.

Well, the inputs I typed were one and two, respectively, for x and y.

So, of course, one is less than two.

So that's why my program printed out, quote unquote, x is less than y.

But recall the code.

The code then proceeded to ask two more questions.

Is x greater than y?

Is x equal equal to y?

Well, the flow chart depicts those questions, too.

Notice that no matter whether the question

had an answer of true or false, the arrows both converge back down

to this second diamond shape here.

And that second diamond shape asks the second question, x greater than y.

That, too, has a true or false answer.

So we go one way or the other.

But if x is one and y is two, then no, the answer is false.

One is not greater than y.

So logically, in the flow chart, you follow the false arrow this time.

And notice, along that false arrow you don't print anything this time.

That's why we only saw one printout on the screen.

Now, there was still a third question.

And this flow chart captures that, as well.

The third diamond asks x equals equals y.

Now that, too, has a false answer in this case, because one, of course,

does not equal equal y.

And so we again follow the third false branch here.

And that leads us, of course, to stop.

And stop just indicates that's it for the program.

So I think that's correct.

And that particular flow chart does happen

to represent the actual code that I wrote.

So it's correct.

It does what it's supposed to do.

It answered the question correctly by printing on the screen x less than y.

But what is, perhaps, poorly designed about it?

Let's make this first distinction.

It's not enough, necessarily, for the code

that you write to be correct and do what you intend.

Longer term, especially as our programs get longer and more sophisticated,

more complicated, we're going to want them to be well-designed, too.

Thoughts on in what way this program is arguably not well designed,

even though it's correct?

Let's see here.

Khalid, if I'm saying that right, your thoughts?

KHALID: Too many ifs, I think, is getting repetitive.

We can make our code more concise, maybe.

DAVID MALAN: Yeah, it seems a little repetitive.

I'm asking if this, if this, if this.

And yet, logically, I should know the answer to some of those later questions

once I figure one out.

And, in short, if you look at this diagram here,

notice that no matter whether I go left or I go right,

I'm always asking three questions.

No matter what, all of those arrows lead to the first, the second,

and the third diamond.

So I'm asking three questions, no matter whether any of those answers

are true or false.

Well how might I go about improving this?

Well, let me propose that we introduce another keyword

to our Python vocabulary, namely elif.

And this, too, is kind of a succinct one.

It's a conjunction of else if, in English,

which allows us to ask a question that takes into account whether or not

a previous question had a true or false answer.

Well, what do I mean by that?

Well, let me go back to my code here.

And let me propose that we now improve upon this, here,

by asking ourselves, ultimately, how can we ask fewer questions?

And let me go ahead here and propose that

instead of asking if, if, if, let's make these conditions potentially mutually

exclusive.

That is to say, don't keep answering questions once we get back

a true answer.

So I'm going to change my code up here as follows.

Instead of asking if, if, if, I'm going to say, if x less than y, elif x

greater than y, elif x equals equals y.

So I'm going to implicitly, just like an English,

take into account that I'm only going to keep asking myself these questions

if I haven't yet gotten a true response.

Think about the logic here, the English.

If x is less than y, on line four, print out x is less than y.

Well, if that's the case, you're done, logically.

Because if the English is saying if x less than y, else if x greater than y,

those are going to be mutually exclusive if the answer to the first question

is true.

You don't have to keep asking questions to which you already logically know

the answer.

So let me go ahead now and run this program.

And I think the behavior is going to be the same.

Python of compare.py, what's x?

Let's do one.

What's y?

Let's do two.

x is less than y.

Now, honestly, I didn't really notice a difference when I ran the program.

And honestly, my Mac, my PC, my phone nowadays,

are so darn fast that these kinds of improvements

aren't going to necessarily feel any faster until we're

writing bigger, faster programs.

But it's laying the foundation for writing better code longer term.

Now what is the improvement I've just made?

Well, if previously my diagram looked like this,

which was problematic insofar as I was asking three questions no matter

what, even if I already figured out what I want to print on the screen.

This new version of the program that says if, elif, elif, might look

a little something like this instead.

Now it got a little wider.

That's just because we drew the arrows to be a bit wider here.

But let's focus on just how many questions are getting asked.

Let me zoom in at the top, as before.

And let me propose that we note that the start oval is at the very top,

and it's asking us to ask one question first.

x less than y, is one less than two?

But notice here, let me zoom out, if one is, indeed, less

than two, we follow this longer arrow down, marked true.

We print out quote, unquote x is less than y.

But then we immediately follow this next arrow down to the icon that says stop.

So that's what's implied by doing if, elif, elif.

If we get back a true answer right away to that first if,

we're going to print out x is less than y and then stop.

We're logically at the end of the program.

So this picture is just representing, graphically,

what the code is actually doing.

But suppose I typed in something else.

Suppose that my code actually ran, and I typed in two for x and one for y.

That is to say, the answer to the first question is now false.

But the answer to the second question is now true.

Because, of course, two is greater than one.

Well, let's go back to the diagram.

Same as before, we start at the very top where it says start.

The very first question up here, now, x less than y, is an answer of false,

because no, two is not less than one.

So we follow this arrow to the next question, this diamond.

Is x greater than y?

Well, yes, two is greater than one.

So now we follow this left arrow, which is true.

We print out quote, unquote x is greater than y, and then stop.

So what's the improvement?

Well, in the first case, we got lucky and we only

had to ask one question and boom, we're done.

This time, we had to ask two questions, but then boom, we're done.

Only if x happens to equal y do we actually find ourselves, logically,

getting all the way down to this final elif in my code.

And pictorially, only if x is equal to y do

we find ourselves going all the way down to the third diamond,

the third question, asking is it equal to y or not?

Now, hopefully, the answer at that point is not false.

We've included a false arrow just so that the program itself

is well-defined.

But, logically, we shouldn't actually be getting there anyway,

because it's got to be less than, or greater than, or equal to in this case.

Well, let me pause here to see if there's

any questions, now, either on the code version thereof here,

or on this diagramming of that very same logic.

Questions here, on this control flow?

SPEAKER 1: Aren't we supposed to put an else at the end?

DAVID MALAN: A good question.

And yes-- so that's going to be my third and final approach.

And if you don't mind, let's pivot there right away.

Identifying a third keyword, that indeed exists

in Python, that allows us to be even better at expressing this logic

to design this program even better.

And that's going to solve a particular problem.

So if I take us back to our code here, notice

that what I've highlighted earlier, elif x equals equals y.

It's not wrong to ask that question.

In fact, if you're trying to be especially thorough,

it makes perfect sense to check if x is less than y, greater than y,

or equal to y.

But why don't I need to ask this third and final question?

SPEAKER 2: We don't need to ask if x is equal to y any more because, logically,

if the two conditionals evaluate to false,

there is only one conditional that will evaluate to true.

And that is x is equal to y.

DAVID MALAN: Exactly.

If we're all pretty comfortable with math, and comparisons

here, of course x is either going to be less than y, greater than y,

or equal to y.

But once you rule out the first two scenarios,

logically, it's got to be the case that x must equal y.

If it wasn't the case, then it's less than or greater than.

So Hope proposed that we use this other keyword, else.

And how do we use this?

Well, exactly as we might in English.

Let me go back to my code here.

And instead of bothering to ask the third and final question,

let's not ask a question at all.

Let's just have this catch-all. so to speak, a final line of code that says,

else just assume that x is equal to y.

Therefore, printing it as well.

So what's the upside of that?

My code is still going to work exactly the same.

And again, my computer is so darn fast, I

don't even notice that it's working even faster than it was before.

But we would notice these kinds of things

if we were doing a lot more work, a lot bigger programs here.

But let me run Python of compare.py.

Let's do, for instance, one and two.

Still works for that.

Let's do two and one.

Still works for that.

Let's do one and one.

And it, indeed, now works for that.

But in these cases now, let's consider the path we just went down.

Previously, our diagram, when we had if, elif, elif in place,

looked a little something like this.

And notice, they began, we might have asked one question, or two,

or worst case, three whole questions.

But we can do better than that, using else, as Hope proposed,

we can whittle this diagram, now, down to this.

And even though it looks like the diagram's getting bigger,

notice that it's having fewer building blocks inside of it.

There's fewer arrows and there's fewer nodes in this picture.

Let's start at the top now.

Start leads us to the first question, still. x less than y?

If the answer is true, great.

We can say as much, x is less than y, and we can stop.

If it's not true, if it's false, we can ask the next question.

x is greater than y, true or false?

If it is, great.

We can print x is greater than y, and stop.

Else, if it's not the case that x is greater than y, the answer is false.

We can just immediately, logically, say x is equal to y.

We don't have to add the third question at all.

We can just immediately conclude there.

So what's the implication here?

You can see, with these pictures, a relative decrease

in the complexity of a program.

The first one was very long and stringy, with lots and lots of questions,

unnecessarily, ultimately.

The next one got a little shorter.

And this one's even shorter still.

And again, the fewer lines of code you have, the less likely

you are, arguably, to make any mistakes.

The easier it is for other people to read.

And so, generally, this readability, this simplification,

is, indeed, a good thing.

Well, let's go ahead and add another piece of capability to Python,

and that's this one here.

Just like in English, where you can ask this question or this other question,

you can say the same thing in Python using literally this word or.

So let me go back to my Python code here.

And let's propose how we might ask a couple of questions

at once this time, perhaps this time considering how we might ask not

whether or not it's greater than or equal to,

and caring about the precise answer.

Let's take a coarser approach here.

And let's just try to determine is x equal to y or not?

Well, let me go ahead and delete some of this code

and change the question we're asking.

Let me do this-- well, if I care about whether it's equal or not,

let's check the possible scenarios.

If x is less than y or x is greater than y, let's go ahead

and print out x is not equal to y.

Now why is that, no pun intended?

If x is less than y, well, it's obviously not equal.

If x is greater than y, it's obviously not equal.

So we can conclude x is not equal to y.

So if we, instead, want to make sure that it is equal to,

we can just use Hope's else, using print quote, unquote x is equal to y.

And again, why is this?

Well, if x is less than y, or x is greater than y,

they're obviously not equal.

Otherwise, logically, they must be equal, in fact.

So let's run this.

Let's go ahead and run Python of compare.py.

What's x?

One.

What's y?

Two.

OK, x is not equal to y.

Let's do it again, put two for x, one for y. x is not equal to y.

And one third time, how about x is one and y is one.

x is now equal to y.

Now if we want to compare that visually, too,

let me propose that the picture looks a little something like this.

And again, this is the exact same thing logically,

but it's a pictorial representation thereof.

What's the first question?

Well, if x is less than y, well, then we follow the true arrow.

And we say quote, unquote x is not equal to y.

And then we stop.

But what if x is not less than y?

What if it's greater than y?

What if it's two and one, respectively?

Then the answer to x less than y, first question, is false.

So we go here.

We ask the second question, because of the or,

and that asks is x greater than y?

If so, notice this, we can kind of reuse some of the same parts of this picture,

and just say x is not equal to y.

We don't need to add arrows and ad boxes unnecessarily.

We can reuse lines of code, parts of the picture, just as we have lines of code.

And then we stop.

Lastly, we have the following.

If we know that x is not less than y, we know

that x is not greater than y, it must be the case that x equals y.

We don't need to ask a third question, another diamond.

We can just immediately print as much, and then say stop, as well.

Well, what could I do here?

I bet I could improve this code slightly.

And if we really want to be nitpicky, I would

argue that this is now really just a minor refinement,

but it's a good habit to get into thinking about.

Could my code be better?

Could my code be simpler?

Could I improve this code further?

It's subtle, but could I improve the design?

Could I ask fewer questions?

Could I tighten it up, so to speak?

What do folks think?

SPEAKER 3: You can ask is x is just equal to y.

Then if you print x is equal to y, else x is not equal to y.

DAVID MALAN: Perfect.

Recall one of the other symbols we saw on the available list earlier.

We can check not just less than, or greater than, or equal to.

We can literally ask the question is it not equal to?

Why are we wasting time asking if it's less than or if it's greater than?

Well, if all you care about is is it not equal, I think we can do exactly that.

Let's just ask the one simple question we do care about.

And so let me go back up here.

And let me just say not both of these questions, let's get rid of the or.

Let's just say if x is not equal to y, then go

ahead and print x is not equal to y.

And that, too, I think is going to work exactly the same.

But the picture now looks a little bit different.

Notice that this was our flow chart earlier,

that represented that same logic.

And there's a bit of complexity.

You've got to go left, you've got to go right,

based on the answer to these couple of questions.

If we now take into account what this version of the program looks like,

it's even simpler, perhaps the simplest one we've seen yet.

When we start off the program, we ask just one, and only one, question,

is x not equal to y?

And if so, true, we go ahead and print out x not equal to y.

If the answer is false, then, of course, it must be equal to y,

so we say that instead.

And if we really want, we could invert this.

If I go back here to my code, and if, for whatever reason,

you just prefer to think in terms of equal or not equal,

as opposed to not equal or equal, it's really up to you.

We could change this to be equals equals.

But I'm going to have to change my print statements to be in the opposite order.

So let me go ahead, now, and reverse these two here,

and move the second one first and the first one second.

So now, when I execute this code, I'm asking still just one question.

So it's still just as good, just as succinct.

But now the diagram, instead of looking like this,

is going to change the not equal to equal equal.

And we just need to make sure that we print out the right thing, accordingly.

And again, here too, just as the code is getting a little more compact, a little

more compact, with fewer and fewer characters,

so are these diagrams, these flow charts capturing the relative simplification

of each of those programs, too.

Let me go ahead and pause here to see if there's any questions, now, on any

of these versions of code.

SPEAKER 4: Yeah, I have a couple of questions.

What if indentation is not used?

DAVID MALAN: If indentation is not used, your program will not work.

So Python is a little different from a lot

of languages in that it enforces the indentation requirement.

Some of you who have been programming for years

might not necessarily be in the best habit of indenting your code properly.

And one of the features, arguably, of Python

is that it makes you indent your code, or it will not just work.

And I think, did you have one other question?

SPEAKER 4: Yeah, is the colon necessary?

DAVID MALAN: Is the colon necessary?

Yes, the colon, too, is necessary.

So with Python, what you see is what you get here.

And, indeed, it needs to be indented and the colon is necessary.

Python does not use, in the same way by convention as C, and C++, and Java,

curly braces to connote blocks.

Instead, it relies, indeed, on this indentation.

Well, let me propose that we introduce one other keyword here in Python,

to see exactly how we might combine additional thoughts.

And that's going to be literally the word and, a conjunction of one,

or two, or more questions that we might want to ask at once.

And let me propose, here, that we explore this kind of logic

by way of another program altogether, in VS Code, whereby I'll go ahead now

and create a new program, say, called grade.py.

Let's consider exactly what grade a student should get,

based on their score on an exam, or a test, or a quiz,

or some other assignment like that.

I'm going to go ahead and run code of grade.py, to give myself a new file.

And I'm going to go ahead and start by just getting the user's score, again,

on some assignment, or test, or the like.

And I'm going to store it in a variable called score, equal the return

value of the int function, which is going to convert whatever the user's

input is when prompted for this score.

So again, the user should just oblige by giving me a number like zero, or one,

or two, or hopefully much higher than that, like 97, 98, 99, 100,

assuming the test or assessment is out of 100 percentage points.

Now, how could I go about assigning a grade to the student's score?

Well in the US, it's very commonly the case

that if you get between a 90 and 100, that's an A.

And if it's between an 80 and a 89, it's a B. If it's 70 and 79, it's a C,

and so forth, all the way down to F, which should be E,

but we'll see that there's a bit of a jump.

So how might I express this?

Well, I can use conditionals.

And I can ask a few questions and then print out the student's grade

accordingly.

So let me express it like this, if the student's score is

greater than or equal to 90, and the student's score is

less than or equal to 100, so it's in that range, let's go ahead

and print out that their grade shall be an A. Because they're in the 90s,

above grades range.

elif the score is greater than or equal to 80,

and the score is less than or equal to, say, 89, but here I have some options.

Logically, I can actually express myself in any number of ways.

And maybe just to be a little cleaner, I'm

going to say a score is less than 90.

So I'm using less than instead of less than or equal to.

So I'm making sure that their boundaries between these grades are correct.

Then, I'm going to go ahead and give the student a B if it's in the 80s.

elif score is greater than or equal to 70, and the score is less than 80,

I'm going to go ahead and give them a C.

elif the score is greater than or equal to 60, and the score is less than 70,

I'm going to go ahead and give him a D. And here's

where it's a little anomalous, at least in some schools here, else

I'm going to go ahead and give them an F. So we're skipping E altogether,

and we're going to give an F, instead, for the grade.

So that's the catch-all.

And I think, logically, I've gotten this correct,

at least based on where I went to school growing up,

such that it's going to give an A, or a B, or a C, or a D,

else it's going to assume that you got an F.

Well, let's try just a few of these here.

Let's run Python of grade.py.

My score is, let's start strong, 100.

I got an A. Didn't do as well the next time, maybe it's a 95--

still an A. Starting to slip further, so I got an 89 the next time.

That's now, say, a B. And let's say I really had a bad week,

and it's now a 71.

That's now a C. Or I didn't even submit it at all, that's an F, altogether.

So it seems to work.

That's not really an exhaustive test, but at least

based on some sampling there, my code seems to work as I expect.

But let's see if we can't tighten this up.

It's not wrong.

It's correct.

And, indeed, according to my own specifications,

I dare say this code is correct.

But can we tighten it up?

Can we reduce the probability of bugs, now or down the line?

Can we increase the readability of it?

And can we increase the efficiency of it?

Can we get the computer to have to answer fewer questions

and still get the same result?

Well, let's see what we might do.

Let me just switch things up, if only to demonstrate that we can

use these symbols in different ways.

I could say, as I've done, if score is greater than or equal to 90.

But I can actually do this, I can flip it around.

Instead of saying greater than or equal to,

let's say 90 is less than or equal to score.

And here, let's say if 80 is less than or equal to score.

And here, 70 is less than or equal to score.

And then, lastly, 60 is less than or equal to score.

So it's the same thing, logically.

I'm just switching things around, just like you could do on paper pencil

if you really wanted.

But now notice this trick.

And this is not possible, for those of you who have programmed in C, or C++,

or Java, or other languages.

Notice what I can do here is actually combine these ranges.

Notice that I'm asking two questions, two Boolean expressions.

Is 90 less than or equal to score, and is score less than or equal to 100?

Well, Python allows you to nest these things like this,

and chain them together.

And just like you would on paper pencil in the real world,

you can encode in Python, do this, which is just a little cleaner.

It's tightening up the code a little bit.

It's fewer keystrokes.

It's faster to type.

It's easier to read, moving forward.

So that's arguably better, as well.

So that's one improvement.

It's largely aesthetic, in this case.

It's still asking the same number of questions,

but it's doing it a little more succinctly still.

Well, what more could I do here next?

Well, you know what?

Each time I'm deciding these grades, I don't

think I have to ask two questions.

I don't have to ask, is it greater than 90 and less than 100?

Is it greater than 80 and less than 90?

If I rethink my logic, I can maybe do this better still.

Let me propose that we simplify this further, and just do this.

If we know that input, for the moment, is going to be within 0 and 100,

we can make some assumptions.

We could say something like, if the score is greater than or equal to 90,

well, the student gets an A. elif the score is greater than or equal to 80,

the student gets a B. elif score is greater than or equal to 70,

they get a C. elif the score is greater than or equal to 60,

they get a D, else they get an F. So what have I done here?

Well, instead of asking two questions every time,

checking the lower bounds and the upper bound of that range,

I'm being a little more clever here by asking if the score is greater than 90,

well, they've obviously gotten an A or better.

If your score is greater than 80, well, you either

deserve an A if it's really strong, or a B if it's just above 80.

But because of the elif logic, we've already checked

is the student's score greater than 90?

And if it's not, then we're asking the question, well, is it greater than 80?

So you implicitly know it's somewhere in the 80 to 89 range,

else you know it's in the 70 to 79 range, else it's in the next range

down.

So it's a minor optimization that allows us to ask fewer questions.

But again, it's making the code, arguably, a little more readable,

certainly more succinct, an then, hopefully, more maintainable

longer term.

Any questions, then, on these types of changes,

and this type of logic with our code?

SPEAKER 4: What if we don't use elif at all?

What if we write the code in if?

DAVID MALAN: Yeah, so that's a good question,

because it's actually going to have an unintended effect here.

Let me get rid of the F temporarily, and just

focus on A through D. If we revert to where

we began today's story, with conditionals, saying if,

if, if, if, now our cleverness here of using broader strokes

and not using an upper and lower bound ranges

is going to come back to be a downside.

Let me go ahead and run Python of grade.py.

And suppose my score is 95.

I am so darn excited.

I want my A, but nope.

I just got an A, a B, a C, and a D. So logically, that's broken things.

Because if you don't make these conditions mutually exclusive,

every one of those questions is going to get asked, and therefore answered.

And even if your grade is above a 90, it's

also, logically, above an 80, above a 70, above a 60,

and if I'd kept it in there, I would have failed, as well, with an F.

Really good question.

Other questions here, on this form of logic?

SPEAKER 5: Would there be any better way to clean up

even just this simple statement, like we had before,

the previous one that you had with the elif?

DAVID MALAN: I like your enthusiasm for simplifying things further.

I'm going to go out on a limb here and say this is about as good as it gets,

at least using only conditional statements.

I can, if my mind wanders, think of a slightly more clever way

to do this, maybe with something called a loop,

or another programming construct.

We don't have that yet in our vocabulary.

But yes, there's absolutely other ways to do it.

But I think not yet if we want to restrict ourselves

to just words like if, and or, and else, and elif, and and, and the like.

Well, let me propose that we pivot now to use another approach here

that uses one other symbol that, up until now,

we've not really had occasion to use.

Let me propose that we implement a program that we'll call parity.

In mathematics, parity can refer to whether a number is even or odd.

And that's kind of an interesting question.

And it turns out it can be useful in other applications,

too, to just ask the question is a given number even or odd,

maybe that the user typed in?

And let me go ahead and write a new program

called parity.py, via code parity.py in my terminal.

And let me propose that we use this as an opportunity

to introduce the last of those arithmetic symbols,

at least most of which we're familiar with, addition, subtraction,

multiplication, division.

But there's been on this list before, this last one here, a percent sign.

And it doesn't mean percentage in this case,

when used as an operator in programming in Python.

Rather, it represents the so-called modulo operator,

for modular arithmetic.

Or, at least in our case, we're going to use it to calculate the remainder when

dividing one number by another.

Well, what do I mean by that?

Well, if you take a number like one divided by three,

three does not go into one cleanly.

So you have a remainder of one.

Two divided by three has a remainder of two.

Three divided by three has a remainder of zero, because it divides cleanly.

Four divided by three has a remainder of one, because you can divide it in once,

but then that leaves one, so it has a remainder of one.

And then lastly, something like five divided by three

has a remainder, of course, of two.

So that's all we mean by remainder, how much is left over

after dividing one number by another.

Well, if I go back now to my code, and I consider how I might implement

the question is this number even or odd?

Let's consider how we might implement that,

since it's perhaps not necessarily obvious how we

can use this additional building block.

But it turns out it's going to be very useful longer term.

Well, let's first just get a number from the user in a variable called x.

And I'm going to set that equal to the conversion to int

of whatever the user inputs, after asking them what's x, question mark.

And we've done that before, many times.

How do I now determine if x is even or odd?

Well, it turns out, if I have access to a programmatic operator that tells me

the remainder, I think I can do this.

In fact, let me just ask the group.

And this is just from grade school math, perhaps,

what does it mean for a number to be even, ?

To be clear, a number like 0, 2, 4, 6, 8, 10, 12, 14, 16,

those are all even numbers.

But what does that really mean?

Elena, if I'm saying that right?

ELENA: Even numbers that can divide it exactly by two.

For example, 2, 4, 6, 8, and 10, and--

DAVID MALAN: Perfect.

And we could go on all day long, literally,

since there's an infinite number of those even numbers.

But it's nice that you formulated it in terms of a question

that we can ask very clearly.

Is this number cleanly divided by two?

That is, can we divide it by two with no remainder, a remainder of zero?

Well, that's perfect, because if we have this operator, this percent sign, that

allows us to answer just that, what is the remainder, we can presumably check

is the remainder zero, or is it one?

Do we have nothing left over, or do we have one left over?

Well, let's ask that.

If x divided by two has a remainder of zero, as Elena proposes, let's go ahead

and print out something like quote, unquote even.

And just say as much to the user.

else, I think we can assume that if a number's not even,

it's going to be odd, if it's, indeed, an integer.

So I'm going to go ahead and print out quote, unquote odd instead.

And let's go ahead and now run Python of parity.py in my prompt.

What's x?

Let's start with two.

Two is, in fact, even.

Let's start with four.

Four is, in fact, even.

Let's get interesting with three.

Three is now odd.

And I think we could do that all day long and hopefully get back, indeed,

exactly that answer.

But what more could we do here?

How could we improve upon this?

Well, recall that we have the ability to invent our own functions.

And let me just propose, for the sake of discussion,

that we're going to eventually find that it's

useful to be able to determine if a number is even or odd.

And so we'd like to have that functionality built-in.

And I don't think Python has a function for telling me just that.

But I can invent it using code like just this.

So let me go into my earlier version here.

And let me propose that we do this.

Let me go ahead and write a main function.

I'm going to get back into that habit of defining a main function to represent

the main part of my program.

And I'm going to do what I did before.

I'm going to get an integer from the user's input,

asking them what's x, question mark.

And then I'm going to ask this question.

For the moment, I'm going to naively assume

that the function already exists, but that's

a useful problem-solving technique.

Even if I have no idea yet where I'm going with this,

how I'm going to invent a function that determines if a number is even,

I'm just going to assume that there's a function called "is even,"

and I'm going to call it, blindly, like this.

If is even, passing in x, then go ahead and print quote, unquote even.

So if this magical function called "is even" returns true, as its return value

I am going to print out that it's even.

Else, otherwise, I'm going to assume that it's, of course, odd.

Now the one problem with this program, even if I call main over here,

is that is even does not exist.

And this program would break if I ran it right now.

But that's OK.

I have the ability, recall, to invent my own function.

So let me define, with def, a function called "is even."

I want this function to take an argument.

And I'm going to call it n, just a number, generically.

I could call it x.

But again, I don't want to confuse myself as to which x is which.

So I'm going to give it a different name, and that's fine.

I'm just going to call it, more generically, n for number.

And then I'm going to do this.

I'm going to say if N % two equals equals zero, just like before, then,

and here's the magic, you, the programmer,

can actually return what are called Boolean values.

We've seen in Python that Python has stirs or strings, ints or integers,

floats or floating point values, all of which

are different types of data in Python.

Python also has a fourth data type called bool for a Boolean value.

And even though this is just adding to our list, the nice thing about bools

is that they can only be true or false.

An int can be any number of an infinite possible values.

A bool can only be true or false.

And it must be capital T and capital F if you're writing itself.

So if I go back now to my code, and I consider

exactly what I want to return here.

Well, if n % two equals equals zero, that is,

if n divided by two has a remainder of zero, well, I think it's even,

to, Elena, your definition.

So let's return true, capital T. else, if it doesn't have a remainder of zero,

I'm pretty sure, mathematically, it's got to have a remainder of one.

But it doesn't matter.

I know it's not even, so I'm going to return false.

And we return false, instead capital F. And now that we've defined both main

and is even, and I'm calling main at the bottom, I think I've got this right.

Python of parity.py, Enter.

What's x?

Let's try something simple, like two.

And it's even.

Let's do it again.

What's x?

How about four?

Even.

Once more, what's x?

How about three?

And it's odd.

Now, what have I done here?

I've just made the point that if I want to create my own function called "is

even," that answers this question for me,

that I can now use, in this program, and heck, maybe future programs

that I write, I now have a function that no one gave me,

I gave myself, that I can use and reuse.

And I can even, perhaps, share it with others.

I'm using that function now on line three, just to make a decision.

I'm using a conditional up there.

And my Boolean expression, something that's true or false,

is going to be not something explicit, like x less than y,

or y greater than x, or the like.

It's going to be a function call.

I'm using a function as my Boolean expression.

But that's OK because I know, because I wrote it,

that that function "is even" returns true or it returns false.

And that's all I need in a conditional to make a decision

to print even or print odd.

So let me pause here to see if there's any questions now on how I've

implemented "is even," using this bool.

SPEAKER 6: Hello, hi David.

First of all, thank you for this wonderful class the day

before yesterday and today, sir.

I have just one query, based on the background of Java.

There, when we used to pass the argument,

we can also pass the address of the variables.

So is there any sort of this concept in Python?

DAVID MALAN: Short answer, no.

Those who are unfamiliar with Java or other languages, or C, or C++,

there's generally ways to pass values in different mechanisms that allow you,

or disallow you, to change them.

In Python, no.

Everything we're going to see is actually, in fact, an object.

But more on that down the line.

How about time for one more question here on these bools and these

"is evens."

SPEAKER 7: So I actually had a question about defining a function,

if that's OK.

DAVID MALAN: Sure.

SPEAKER 7: So if you define one, within your code, like you made it up,

are you allowed to use the dot operator like we did name dot strip,

and use it like that?

DAVID MALAN: Good question.

If you've created your own function, can you

use other functions, like dot strip, or dot title, or dot capitalize,

that we've seen in the past?

You can use those on strings.

Those functions come with strings.

You can't necessarily use them on your own functions,

unless your function returns a string, for the examples you gave.

I'm returning a bool.

Bools have no notion of white space to the left or the right.

You can't call strip, you can't call capitalize.

But if you were writing a different function

that returns a string, absolutely.

You could use those functions, as well.

Well, let me turn our attention, if I may, back to this example

here, and consider, as we now frequently do,

can we improve on the design of this code?

Can I make this particular program better?

And I can.

There's a couple of ways here.

And I'll show you something that's now generally known as something Pythonic.

There's actually this term of art, in the Python world,

where something is Pythonic if it's just the way you do things in Python.

Which is to say, we've seen already there's

so many different ways to solve certain problems.

And in the Python community of programmers,

there tend to be some ways that are smiled upon more than others.

And they tend to relate to features that maybe only Python has,

but not other languages.

And here's some syntax that you might not have seen in languages like Java,

or C, or C++ if you've programmed before.

And if you've never programmed before, this too is going to be new.

Instead of asking a question like this, if else using four lines, in Python,

you can actually collapse this into just one more elegant line, if you will.

Instead of asking if n divided by two has a remainder of zero,

return true, else return false.

Let me delete all of that and just say this, return true if n divided by two

has a remainder of zero, else return false.

Now those of you who do have prior programming experience

might actually think this is kind of cool.

You can condense, from four lines into one line, that very same thought.

And one of the reasons why Python is popular is that it does

tend to read rather like English.

It's not quite as user-friendly as most English, or most human languages.

But notice, now, the line does rather say what you mean.

Return true if n divided by two has a remainder of zero, else false.

That's pretty darn close to something you might say, logically, in English,

be it about even and odd or really anything else.

So that program is going to work exactly the same.

Python of parity.py, let me type in two.

It's still even.

Let me type in three.

It's still odd.

But I can refine this even further.

And again, consistent with this idea of not just writing correct code,

but writing better and better code, but still keeping it readable,

I can do one even better than this.

Notice this value here is my Boolean expression.

And it is going to evaluate to true or false.

Is n divided by two having a remainder of zero or not?

That is, by definition, a Boolean expression.

It has a yes/no answer, a true/false answer.

Well, if your Boolean expression itself has a true or false answer,

why are you asking a question in the first place?

Why ask if?

Why say else?

Just return the value of your own Boolean expression.

And perhaps the tightest version, the most succinct, and still readable,

version of this code would be to delete this whole line, Pythonic

though it is, and just return n modulo two equals equals zero.

If it helps, let me add parentheses temporarily,

because what's going to happen in parentheses will happen first.

n divided by two either does or does not have a remainder of zero.

If it does, the answer is true.

If it doesn't, the answer is false.

So just return the question, if you will.

You don't need to wrap it, explicitly, with an if and an else.

And in fact, because of order of operations,

you don't even need the parentheses.

So now this is perhaps the most elegant way to implement this same idea.

Now, which is better?

This is pretty darn good.

And it's hard to take fault with this because it's so very succinct.

But it's perfectly OK, and just as correct,

to have an if and then an else.

Even though it might be four total lines, if that helps

you think about your code more clearly, and it helps

other people reason about it, as well.

So it turns out there's another syntax that you

can use to implement the same idea of a conditional,

whereby you do something optionally, based on the answer to some Boolean

expression.

And the keyword that you can now use, in recent versions of Python,

is called this-- match.

Match is a mechanism that, if you've programmed before, is similar in spirit

to something called switch in other languages.

For instance, let me go ahead here and close out parity.py And let me go ahead

and create a new file called house.py.

And in house.py, I think what we're going

to do is try to implement a program that prompts the user for their name,

and then just outputs what house they're known

to be in in the world of Harry Potter.

So for instance, let me go ahead and do this.

Let me give myself a variable called name, set it equal to the return

value of the input function.

And I'll say something like, what's your name, question mark.

And then after that, I'm just going to use

a traditional if, elif, else construct to decide what house this person is in.

So let me say if name equals equals, say Harry, as in Harry Potter, well,

let's go ahead and print out Harry's house, which is Gryffindor

in the world of Harry Potter.

elif the name is, instead, Hermione, then

go ahead and print out also quote, unquote Gryffindor,

as she's in the same house, too.

elif name equals equals Ron, let's go ahead

and similarly print out Gryffindor quote, unquote.

And let's make this a little more interesting now.

elif name equals quote, unquote how about Draco?

Draco Malfoy, in the books-- let's go ahead and print out quote,

unquote Slytherin.

And just in case someone else's name gets inputted,

for now, let's just suppose that we don't recognize them,

and say, by default, else print out quote,

unquote who, question mark, just to convey

that we don't actually have a hard-coded response to that particular name.

Let me go ahead, now, and run this as Python of house.py, Enter.

And I'll go ahead and type in something like Harry.

And voila, we see that Harry is, indeed, in Gryffindor.

Let's run it one more time, Python of house.py.

Let's type in Draco this time.

Slytherin.

And now, let's type in an unrecognized name.

Let's go ahead and rerun Python of house.py.

And let's go ahead and type in Padma, Enter.

And who?

Because we haven't actually hard-coded with an elif condition in this case,

what house Padma is meant to be in.

Well, it turns out there's other ways to implement this.

Indeed, there's some redundancy here, in that

we're checking if Harry, or Hermione, or Ron are all in Gryffindor.

I feel like we can at least tighten this code up a little bit,

using techniques we've seen already.

So let me go ahead and do this.

Let me go up here and instead do something like this.

Let's get rid of these two blocks of elifs,

leaving just Harry's for a moment.

And let's use that "or" keyword again, and say or name

equals equals quote, unquote Hermione, or name equals quote, unquote Ron,

thereby consolidating all three cases, if you will, into just one

if statement.

Then we still have a separate elif for Draco because he's not,

in fact, in Gryffindor.

And then the final else to catch anyone else.

Let me go ahead now and run this version of the program, Python of house.py.

I'll type in Hermione this time.

She, too, is still in Gryffindor.

Let me try it with Ron.

And that, too, still seems to be correct.

Well, it turns out there's another approach altogether that can perhaps

make your code a little less verbose.

You could imagine how complicated this code might

get if we had not just Harry, and Hermione, and Ron, but a whole bunch

of other names as well, for Gryffindor, for Slytherin, and for all

of the other Hogwarts houses.

So you could imagine that code just getting pretty unwieldy pretty fast.

Well, it turns out another technique you can

use is, indeed, this keyword called match, which is very similar in spirit,

but the syntax is different.

And it allows you to express the same ideas a little more compactly.

So let me go back to house.py.

And let me propose that I get rid of my current if, elif, else approach,

and instead do this.

Literally use the keyword match, and type the name of the variable,

or value, that we want to match on.

And then I'm going to go ahead and include a colon.

And then underneath that, I'm going to include, literally,

a keyword called case.

And the first case I want to consider is going to be Harry.

And I'm going to put Harry in quotes, because it's a string or a stir.

And I'm going to have another colon at the end of this line.

And indented under that one, I'm going to go ahead and, for now,

print out Gryffindor, which, of course, is Harry's house.

Otherwise, I'm going to have another case for quote, unquote Hermione.

And similarly, I'm going to have under that indented,

print quote, unquote Gryffindor, close quote.

Now I'm going to have another case for Ron, also in quotes, with a colon.

Now print quote, unquote Gryffindor.

And now I'm going to have a other case for, let's say, Draco.

This one gets a little more interesting because Draco, of course,

now is in Slytherin.

And then I'm going to go ahead and leave it as that for now.

So let me go ahead and save this file, and go back down to my terminal window,

running Python of house.py, Enter.

And let's go ahead and try Harry.

And he seems still to be in Gryffindor.

Let's run it again for Hermione, Enter.

Gryffindor.

Let's skip ahead to Draco, and type in Draco's name.

He is, indeed, in Slytherin.

Now let's try another name that we haven't handled

a case for, like Padma again, Enter.

And we're just ignored.

There's no output whatsoever because there wasn't a case for Padma.

Now we could, of course, go back in and explicitly add one for Padma.

But what if we, similarly to the else construct,

just want a catchall that handles anyone whose name is not explicitly specified?

Well, turns out the syntax for that, using this new match statement,

is to still have another case, but then to use

this single underscore character, which is used in other contexts in Python.

But for here, it's meant to say whatever case has not yet been handled,

go ahead and print out, as we did before, for instance,

quote, unquote who, with a question mark at the end.

Now let's go ahead and rerun this Python of house.py.

I'll type Padma's name again.

And this time, I think we're at least going

to get an explicit response indicating who,

whereas previously we did not have the equivalent of that.

Now, I think we've regressed a little bit.

We went from tightening things up by putting Harry, and Hermione,

and Ron all on the same line in the same if statement.

But here, we have now three case statements, again, for all three

of those.

Well, we can tighten this code up, as well.

But the syntax is going to be a little bit different.

I'm going to go ahead and delete these two middle cases for Hermione and Ron.

And then up here, next to Harry's name, before the colon,

I'm going to go ahead and use a single vertical bar, and then

a quote, unquote Hermione.

Then another single bar and do quote, unquote Ron.

And this is how, using this relatively new match statement,

you can say the equivalent of Harry, or Hermione, or Ron,

but more concisely than you could using an if statement

alone, as we implemented it previously.

So now, one final run of the program with Python of house.py.

Let's make sure that Harry is still in Gryffindor.

Let's make sure that Hermione is still in Gryffindor.

Let's make sure that Ron is still in Gryffindor.

And indeed, all three of them are.

Now, as always with Python and programming more generally,

there's going to be different ways you can solve these problems.

This is just another tool in your toolkit.

Arguably, it has tightened things up.

Arguably, it's perhaps a little more readable

because there's a little less syntax going on,

a little less duplication of equal signs and elif,

and elif, and elif all over the place.

But ultimately, this would be an equally correct approach to that same problem.

But it turns out with a match statement you

can do even more powerful forms of matching, as well.

Here, we've used it simply to implement the same idea as that if, elif,

else construct.

And it's worth noting, if you've programmed in some other language,

the syntax here is, indeed, correct.

You do not need, for instance, a break statement,

as has been peppered throughout.

And you don't need something like default, or something explicit.

You, indeed, just use this underscore as your catchall at the end of the match.

So just by adding in some of these new keywords

here, like if, and elif, and else, we have now the ability

to ask questions about values.

We have the ability to analyze input from users,

and ultimately make decisions about it.

And these, then, where our conditionals.

Lying ahead is going to be the ability for us to not only use functions,

and variables, and also these conditionals,

but also, next, loops-- the ability to do something, now, again and again.